



SYSTEM AND METHOD FOR  
WEB-BASED REMOTE PROCEDURE CALL (RPC)

CROSS-REFERENCE TO RELATED APPLICATIONS

5 Not Applicable

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR  
DEVELOPMENT

Not Applicable

REFERENCE TO A MICROFICHE APPENDIX

Not Applicable

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to transmitting information over computer networks, and more particularly, to a system and method for executing a remote procedure call (RPC) from a Web page displayed in a Web browser to a server.

2. Description of the Related Art

The Internet is a computer network that provides access to the World Wide Web ("the Web"); a vast collection of pages comprised of text, hyperlinks, graphical elements, input elements (e.g. HTML text box), and program code (e.g. JavaScript). Graphical user

interface programs called Web browsers are employed by Internet users to receive, or download, the Web pages from servers and display the pages at their client devices. A Web browser displays Web pages by interpreting the HTML document and executing any program code, which results in the showing text, hyperlinks, graphical elements, and  
5 input elements on a client display screen.

The rapid increase in the number of Internet users and the ubiquity of the Web browser have prompted companies to adopt Web pages for delivering, to end-users, time-sensitive information and mission-critical software applications. These Web-based applications generally require numerous transmissions of data between a server and the  
10 Web browser. Traditionally, for each transmission of data, a new page must be transmitted back to the user or their existing page must be refreshed. This traditional data transmission model, when used in the context of delivering web-based applications to end-users, greatly increases the number of redundant Web page refreshes. Web page refreshes are problematic, because they result in unnecessary server and bandwidth  
15 utilization, and force the user to wait for redundant pages to load. This negatively impacts the Web-based application's effectiveness and the Internet user's productivity, while compromising network efficiency and scalability.

Several developments illustrate the great effort at reducing redundant page refreshes and for providing a rich interactive user experience for Web pages without  
20 utilizing Web browser plug-ins, such as Macromedia's Flash player or Microsoft's ActiveX controls. One of which was developed and promoted by Microsoft.

Microsoft developed a Java Applet-based technology called "Remote Scripting", which allows information to be exchanged between a displayed Web page and the server,

without having to refresh the page. After several years of promotion by Microsoft, however, "Remote Scripting" technology has not been widely adopted by Web developers. There are several reasons for this:

1. Slow: In order to use "Remote Scripting," each Web page must include a Java Applet, acting as the client-side communication proxy, which must initialize Java Virtual Machine at the client's computer. Initialization of Java Virtual Machine and loading of the Java Applet is a slow process that introduces an undesirable delay in displaying the Web page for most personal computers.
2. Limited Server Compatibility: "Remote Scripting" only supports Microsoft Web server software, such as Microsoft's IIS (Internet Information Server). Most high traffic and established commercial Websites, however, use a non-Microsoft platform or wish not to be limited to only using the Microsoft platform in the future.

As a result of these serious limitations, most Web-based applications do not or cannot take advantage of Microsoft's "Remote Scripting" technology.

Consequently, the most common method for achieving real-time Web page interactivity today, without Web browser plug-ins, is by using a `<iframe>` element. The `<iframe>` element serves as the communication proxy for the Web browser. Data is transmitted to the server by utilizing a `<form>` element and either the "post" or "get" method. Alternatively, one could transmit data to a Web page without utilizing the `<form>` element by dynamically changing the "src" attribute of an `<iframe>` element at the Web browser. The `<iframe>` method for data transmission overcomes the limitations

of Microsoft's "Remote Scripting," but introduces one new serious limitation, which greatly compromises the operation of Microsoft's Web browser.

Specifically, each time new information is loaded into an <iframe> element, a record or URL (Universal Resource Locator) is added to the Microsoft Web browser's history list. That is, the browser is now adding every action performed from the Web page to the history list, rather than just each unique Web page. Thus, since each unique Web page may make several data transmissions, several history records are now being associated with each unique Web page. This is problematic in the context of Web-based applications for the following reasons:

1. Clutters the history list: users must navigate through a large directory tree full of redundant entries before finding the desired Web page.
2. Compromises Web page navigation: users must repeatedly click on the "Back" or "Forward" buttons before being able to leave the current Web page.
3. Produces undesirable application operations: users will trigger duplicate operations at the server (e.g. charge my credit card) when navigating with the browser's "Back" and "Forward" buttons.

These problems affect many Internet users today as the majority of these users utilize a Microsoft Web browser. Furthermore, as Netscape winds down its Web browser business and Microsoft becomes the de facto Web browser, this problem will soon plague every single Internet user, and thus cannot be ignored.

Consequently, NetGratus Inc. developed a method for transmitting data to and from a server by utilizing the <script> element as the communication vehicle. This method prevents unwanted history records from being added to the browser's history list,

however, introduced a new limitation. When data is transmitted via the <script> element, the Web page is limited in the amount of data it can transmit to a server. This is due to the fact that the <script> element utilizes the “get” method to transmit data. If the “post” method could also be utilized to transmit data, however, data could be transmitted to and  
5 from a displayed Web page without limitation of size.

From the discussion above, it should be apparent that there is a need for better method of transmitting data between a displayed Web page and the server over HTTP. Specifically, a method that offers at least the same benefits of the <iframe> method, without disrupting the Microsoft Web browser’s history list. The present invention  
10 fulfills this need.

#### BRIEF SUMMARY OF THE INVENTION

The invention permits a Web page to make JavaScript-based remote procedure calls (RPC) from Microsoft Web browsers to a server by using the HTML <form> and  
15 <iframe> elements, without adding a record (or Universal Resource Locator, URL) to the browser’s history list. The Web page essentially uses HTML <form> and <iframe> elements as the communication vehicle for sending and receiving data to and from the Web server. As part of this invention, it enables submission of data to a server such that the response is no longer delivered onto a new HTML page, but rather to a hidden HTML  
20 <iframe> element.

This invention enables web developers to create web-based applications that utilize a pure-Internet architecture (i.e. where the thin client utilizes only JavaScript and HTML) that yield a traditional fat client application experience, without all the weight

and limitations of the fat client. This is accomplished by, returning new information from server-side business logic to the displayed HTML page, and by using JavaScript/DHTML to alter the presentation layer accordingly. More specifically, this invention allows for data to be submitted to a particular procedure residing at a server, and return the response  
5 from that procedure to the displayed Web page over HTTP.

This invention enables these interactions between the Web page and server by utilizing a client-side proxy, the client's communication agent, and a server-side proxy, the server's communication agent. The client-side proxy is ultra-fast and ultra-light since it is built using 100% JavaScript and HTML (<form> and <iframe> elements)  
10 technology. The server-side proxy can be implemented on virtually any platform, such as dynamic server pages (e.g. ASP, PHP, JSP, CFM, etc.), ISAPI-based C/C++, Java technologies, and so on.

#### BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

15 FIGURE 1 shows a system having a Web page on client computer and Web server constructed in accordance with the present invention.

#### DETAILED DESCRIPTION OF THE INVENTION

Typically, a Web page sends data from a web page to the server when an HTML  
20 form is submitted. This usually occurs when the user clicks the submit button of the HTML form. Once the submit button has been clicked, the browser collects the values from the form fields, builds an HTTP request, and then sends that request to the server. The server receives the information, does what it needs to do – querying a database for

example – and then returns the result of the form submission, usually in the form of a new HTML page.

This invention uses the HTML <form> element in conjunction with a hidden HTML <iframe> element to yield completely different results than previously. Upon a trigger event, such as a mouse-over or a button click, the Web page now places a remote procedure call (RPC) through a client-side proxy, and will receive the results in a hidden <iframe> element, such that the displayed page is no longer refreshed. Additionally, using the <iframe> element to receive the results will no longer add a record of the RPC or URL to the browser's history list.

Once the RPC has been triggered, the client-side proxy includes the necessary parameters for executing the RPC, including the location and name of the remote procedure, the data being passed to the remote procedure, the callback function name, etc. The client-side proxy must also prepare to receive the returned data. It dynamically generates a hidden HTML <iframe> element at the Web browser, and sets the <form> element's "target" property to the <iframe> element.

Now that the client-side proxy is prepared to send the request and receive the response, the <form> element is used to send the request to the Web server. Although the "post" or "get" request method can be used to submit the form; this invention utilizes the "post" method so that the data submitted to the server may exceed 1 kilobyte in size.

Once the HTTP request arrives at the Web server it is passed to the server-side proxy. First, the server-side proxy parses the HTTP request to extract the necessary parameters. Then it regenerates the RPC as if it were a local procedure call. After the call has dispatched, the server-side proxy receives the returned data from the procedure,

packages that data in a format, which is understood by the client-side proxy, and returns that data package or response via HTTP to the <iframe> element.

When the data is received by the <iframe> element, a callback function is invoked to process the returned data. The callback function is a JavaScript function of the Web page, which will process the returned data. It may display the data to the user, or use it to carry out another task, such as performing another calculation (at the client or server). This entire process is accomplished without having to refresh the page.

Now, in order to avoid adding a URL to the browser's history list, a given <iframe> element can only be used once per RPC. Therefore, for each new RPC, the client-side proxy generates a new <iframe> element at the Web browser, and deletes the previous instance of the <iframe> element, as if the previous <iframe> element had never existed. Although a new <iframe> must be used for each RPC, the same <form> element may be used over and over. This unique practice and method for disposing of the <iframe> element is a key aspect of this invention, which greatly increases the usability of a Microsoft Web browser. Specifically, it prevents the history list from becoming cluttered with frivolous records or URLs associated with each RPC, and more importantly, it maintains the usability of the browser's "back" and "forward" buttons for navigating between Web pages.

## 20 System Configuration

FIGURE 1 shows a system 100 having a Web page on client's computer 101 and one Web server 107. Web page 101 includes a client-side proxy 102, which contains HTML <form> 103 and <iframe> 104 elements. Client-side proxy 102 can communicate



with the Web server 107. The client-side proxy 102 assembles the HTTP request and sends it to the Web server via <form> element 103. The Web server utilizes server-side proxy 108, which handles HTTP the requests from client-side proxy 102 and dispatches the procedure calls to various server procedures 109. Server procedures 109 can access  
5 data sources, such as database 110, and return those results to server-side proxy 108. The server-side proxy 108 then packages the results into a format understandable by the client-side proxy 102, and returns that data package to the client-side proxy. The <iframe> element 104 at the client-side proxy 102 then accepts the data returned from the server-side proxy 108. Once this data is loaded into the <iframe> element 104, it is  
10 accessible to client-side processes and a callback function is triggered at the client-side proxy 102 to process returned data.

### *System Implementation*

The next section shows a sample code for implementing the client-side proxy 102.  
15 The received data is stored in the <iframe> element until the new <iframe> element is generated by the next RPC. The <iframe> element is only used once to prevent adding a record of the RPC or URL to the Web browser's history. The code below is taken from the sample Web page "client.htm."

```
20      <html>
          <script>
              var ifrCount=0
              function _I(callback, param)
              {
25                  window.status="";
                  eval(callback + "(" + param + ")");
              }

              function LMLPost(serverPage, callback, serverFunction, data)
              {
```

```

var strHTML=new String();
if(!document.forms.LMLform)
{
    strHTML="<div><form name=\"LMLform\"
5         method=\"post\" action=\"\"+ serverPage + "\">";
    strHTML+="<input type=\"hidden\" name=\"func\">";
    strHTML+="<input type=\"hidden\" name=\"params\">";
    strHTML+="<input type=\"hidden\" name=\"callback\">";
10    strHTML+="</form></div>";
    document.body.insertAdjacentHTML("BeforeEnd",strHTML)
}

if(ifrCount>0)
15    document.all('div_'+(ifrCount-1)).innerHTML="";
    document.forms.LMLform.func.value = serverFunction;
    document.forms.LMLform.params.value = data;
    document.forms.LMLform.callback.value = callback;
    strHTML="<div id='div_'+ifrCount+'><iframe
20        style=\"position:absolute;visibility:hidden;left:0;top:0\"
        name = \" + ifrCount + \"></iframe></div>";
    document.body.insertAdjacentHTML("BeforeEnd",strHTML);
    document.forms.LMLform.target=" "+ifrCount;
    document.forms.LMLform.submit();
    ifrCount++;
25 }

function post_data()
{
    LMLPost("hello.asp", "my_callback", "getData",
30    document.all('input_buffer').value);
}

function my_callback(data)
{
35    document.all('output_buffer').value = data;
}
</script>
<body>
40    <textarea name='input_buffer' rows=8></textarea>
    <textarea name='output_buffer' rows=8></textarea>
    <input type='button' value='Post' onclick="post_data();">
    </body>
</html>

```

45 As shown in the above sample codes for both Web page 101 and the client-side proxy 102, the client-side proxy 102 uses a <form> element 103 as the postman to send data to the Web server and uses <iframe> element 104 as the receiver to receive the returned data from the server proxy 108. This invention uses the <form> and <iframe>

elements together in a combined new and innovated way to transmit data between the displayed Web page and a server, and more specifically to make an RPC, without adding a URL to the Web browser's history list.

Each time a Web page makes a RPC, the client-side proxy 102 generates an  
5 HTTP request that includes the location and name of the remote procedure, any data to be passed to the remote procedure, and the callback function name. This HTTP request is posted to Web server 107 through the submission of <form> element 103. The response arrives to <iframe> element 104 and triggers the specified callback function, so that Web page 101 can process the returned data.

10 The server-side proxy 108 can be implemented by any Web server-side technologies, such as, ASP, PHP, JSP, EJB, ISAPI, and so on. The next section shows an implementation of server-side JavaScript code running on Microsoft's IIS Web server. It can be easily ported to other Web server-side technologies. The following code is taken from the server proxy file "serverproxy.inc."

```
15 <%@ Language=JavaScript %>
    <%
        var _LML_PARAMS_DELIM = String.fromCharCode(1);
        var i, strBody, strResult;
        var strFunc, strCallback;
20        var strParams, arrParams;
        strFunc = Request("func").item;
        strParams = Request("params").item;
        strCallback = Request("callback").item;
        arrParams = strParams.split(_LML_PARAMS_DELIM);
25        strParams = "";
        for (i = 0; i < arrParams.length; i++)
        {
            if (i > 0)
                strParams += ",";
30            strParams += "\"" + EscapeSpecialChars(arrParams[i]) + "\"";
        }
        strResult = escape(eval(strFunc + "(" + strParams + ")"));
        strBody = "<HTML><HEAD></HEAD>"
        strBody += "<BODY onload=parent.window._I(";
35        strBody += "\"" + strCallback + "\",";
```

```

5      strBody += "\"" + strResult + "\"";
      strBody += ">";
      strBody += "</BODY></HTML>";
      Response.Write(strBody);
      function EscapeSpecialChars(strSrc)
      {
          var strDest = strSrc;
          strDest = strDest.replace(/\\/g, '\\\\');
          strDest = strDest.replace(/\r/g, '\r');
10      strDest = strDest.replace(/\n/g, '\n');
          strDest = strDest.replace(/"/g, "\"");
          strDest = strDest.replace(/'/g, "'");
          return strDest;
      }
15      %>

```

The next section shows the sample code for server function 109. This is an implementation for Microsoft IIS and ASP. It can be easily ported to other Web server-side technologies. The first line is used to include the source code of server proxy 108.

20 The simple "getData" function herein returns whatever data it takes as the input argument. The code below is taken from the sample server page "server.asp."

```

25      <!--#include file="serverproxy.inc"-->
      <script runat=Server language=VBScript>
          function getData(strPara)
              getData = strPara
          end function
      </script>

```

The above sample codes illustrate a sequential RPC process. By using additional

30 <form> and <iframe> elements, however, the client-side proxy 102 can make multiple simultaneous RPCs. Although the source code above enables a Web page to directly access server-side procedures through the "post" request method, with little modification, it can also enable a Web page to directly access services provided by server-side objects through the "get" request method.

35 The present invention has been described above in terms of presently preferred embodiments so that an understanding of the present invention can be conveyed. There

are, however, many configurations for network data delivery not specifically described herein, but with which the present invention is applicable. The present invention should therefore not be seen as limited to the particular embodiments described herein, but rather, it should be understood that the present invention has wide applicability with respect to network data delivery generally. All modifications, variations, or equivalent arrangements and implementations that are within the scope of the attached claims should therefore be considered within the scope of the invention.

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000